Sales Forecasting

**Problem Statement:**

Fresh Analytics aims to forecast the demand for various items across multiple restaurants. Accurate sales predictions enable better decision-making around staffing, inventory, and pricing strategies.

**Datasets Used:**

1. **restaurants.csv**: Details about restaurants (id, name)
2. **items.csv**: Details about items (id, name, kcal, cost, store\_id)
3. **sales.csv**: Sales data (date, item, price, item\_count)

**Project Tasks:**

**1. Preliminary Analysis**

* Load and inspect datasets.
* Handle missing values and outliers.
* Merge all datasets to a single DataFrame.

**2. Exploratory Data Analysis (EDA)**

* Analyze overall and time-based sales patterns.
* Weekday, monthly, and quarterly trends.
* Restaurant-wise and item-wise comparisons.
* High-performing restaurants and items.
* Caloric and cost analysis of top items.

**3. Feature Engineering**

* Extract temporal features: year, month, day, weekday, quarter
* Aggregate sales on different levels.

**4. Model Building and Forecasting**

* Models: Linear Regression, Random Forest, XGBoost.
* Train on historical data, test on the last 6 months.
* Evaluate using RMSE.
* Forecast for next year using the best model.

**Insights:**

* Identified top-selling items and restaurants.
* Analyzed temporal patterns in demand.
* Predicted future sales using ML algorithms.

**Source Code:**

# 1. Import Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from datetime import datetime

from sklearn.linear\_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor

from xgboost import XGBRegressor

from sklearn.metrics import mean\_squared\_error

from sklearn.model\_selection import train\_test\_split

# 2. Load Datasets

df\_rest = pd.read\_csv("resturants.csv")

df\_items = pd.read\_csv("items.csv")

df\_sales = pd.read\_csv("sales.csv")

# 3. Rename columns to avoid conflicts

df\_items.rename(columns={'name': 'item\_name'}, inplace=True)

df\_rest.rename(columns={'name': 'store\_name'}, inplace=True)

df\_sales.rename(columns={'name': 'item\_name'}, inplace=True)

# Optional: View column names

print("Sales Columns:", df\_sales.columns)

print("Items Columns:", df\_items.columns)

print("Restaurants Columns:", df\_rest.columns)

# 4. Merge Datasets

df = df\_sales.merge(df\_items, on='item\_name', how='left') \

.merge(df\_rest, left\_on="store\_id", right\_on="id", suffixes=('\_item', '\_store'))

# 5. Convert 'date' and drop missing

df['date'] = pd.to\_datetime(df['date'])

df.dropna(inplace=True)

# 6. Feature Engineering

df['year'] = df['date'].dt.year

df['month'] = df['date'].dt.month

df['day'] = df['date'].dt.day

df['weekday'] = df['date'].dt.dayofweek

df['quarter'] = df['date'].dt.quarter

# 7. EDA

# Overall sales trend

plt.figure(figsize=(12,5))

df.groupby('date')['item\_count'].sum().plot(title="Daily Sales Trend")

plt.xlabel("Date")

plt.ylabel("Total Items Sold")

plt.tight\_layout()

plt.show()

# Sales by weekday

plt.figure(figsize=(8,4))

sns.barplot(data=df, x='weekday', y='item\_count', estimator=sum)

plt.title("Sales by Weekday")

plt.xlabel("Day of Week (0=Monday)")

plt.ylabel("Total Items Sold")

plt.tight\_layout()

plt.show()

# Top 10 items

top\_items = df.groupby('item\_name')['item\_count'].sum().sort\_values(ascending=False).head(10)

print("Top 10 Items Sold:\n", top\_items)

# Most profitable store

df['revenue'] = df['item\_count'] \* df['Price']

top\_store = df.groupby('store\_name')['revenue'].sum().sort\_values(ascending=False)

print("Most Profitable Stores:\n", top\_store)

# Most expensive item per store

expensive\_items = df.groupby('store\_name').apply(lambda x: x.sort\_values('cost', ascending=False).iloc[0])

print("Most Expensive Item per Store:\n", expensive\_items[['item\_name', 'cost', 'kcal']])

# 8. Modeling and Forecasting

# Aggregate daily sales

daily\_sales = df.groupby('date')['item\_count'].sum().reset\_index()

daily\_sales['year'] = daily\_sales['date'].dt.year

daily\_sales['month'] = daily\_sales['date'].dt.month

daily\_sales['day'] = daily\_sales['date'].dt.day

daily\_sales['weekday'] = daily\_sales['date'].dt.dayofweek

daily\_sales['quarter'] = daily\_sales['date'].dt.quarter

# Train/Test Split: Last 6 months as test

cutoff = daily\_sales['date'].max() - pd.DateOffset(months=6)

train = daily\_sales[daily\_sales['date'] <= cutoff]

test = daily\_sales[daily\_sales['date'] > cutoff]

features = ['year', 'month', 'day', 'weekday', 'quarter']

X\_train = train[features]

y\_train = train['item\_count']

X\_test = test[features]

y\_test = test['item\_count']

# Models

models = {

"Linear Regression": LinearRegression(),

"Random Forest": RandomForestRegressor(n\_estimators=100, random\_state=42),

"XGBoost": XGBRegressor(n\_estimators=100, random\_state=42)

}

print("\nModel Evaluation (RMSE):")

for name, model in models.items():

model.fit(X\_train, y\_train)

preds = model.predict(X\_test)

rmse = mean\_squared\_error(y\_test, preds, squared=False)

print(f"{name}: RMSE = {rmse:.2f}")

# Use best model (assume Random Forest)

best\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)

best\_model.fit(X\_train, y\_train)

# Forecast next 365 days

future\_dates = pd.date\_range(start=daily\_sales['date'].max() + pd.Timedelta(days=1), periods=365)

future\_df = pd.DataFrame({'date': future\_dates})

future\_df['year'] = future\_df['date'].dt.year

future\_df['month'] = future\_df['date'].dt.month

future\_df['day'] = future\_df['date'].dt.day

future\_df['weekday'] = future\_df['date'].dt.dayofweek

future\_df['quarter'] = future\_df['date'].dt.quarter

# Predict

future\_df['predicted\_sales'] = best\_model.predict(future\_df[features])

# Plot historical + future forecast

plt.figure(figsize=(14,6))

plt.plot(daily\_sales['date'], daily\_sales['item\_count'], label="Historical Sales")

plt.plot(future\_df['date'], future\_df['predicted\_sales'], label="Forecasted Sales", linestyle='--')

plt.title("Sales Forecast for Next Year")

plt.xlabel("Date")

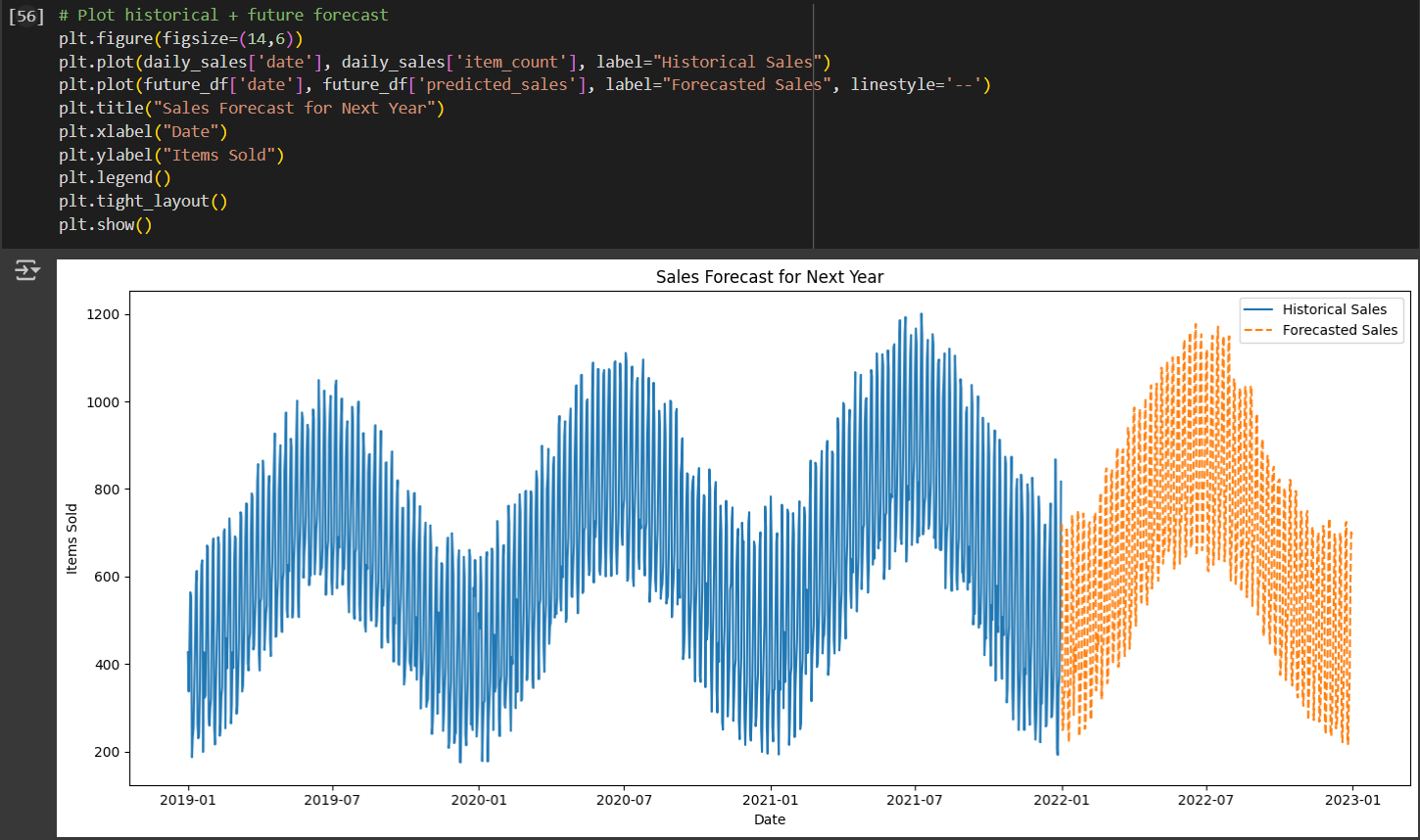
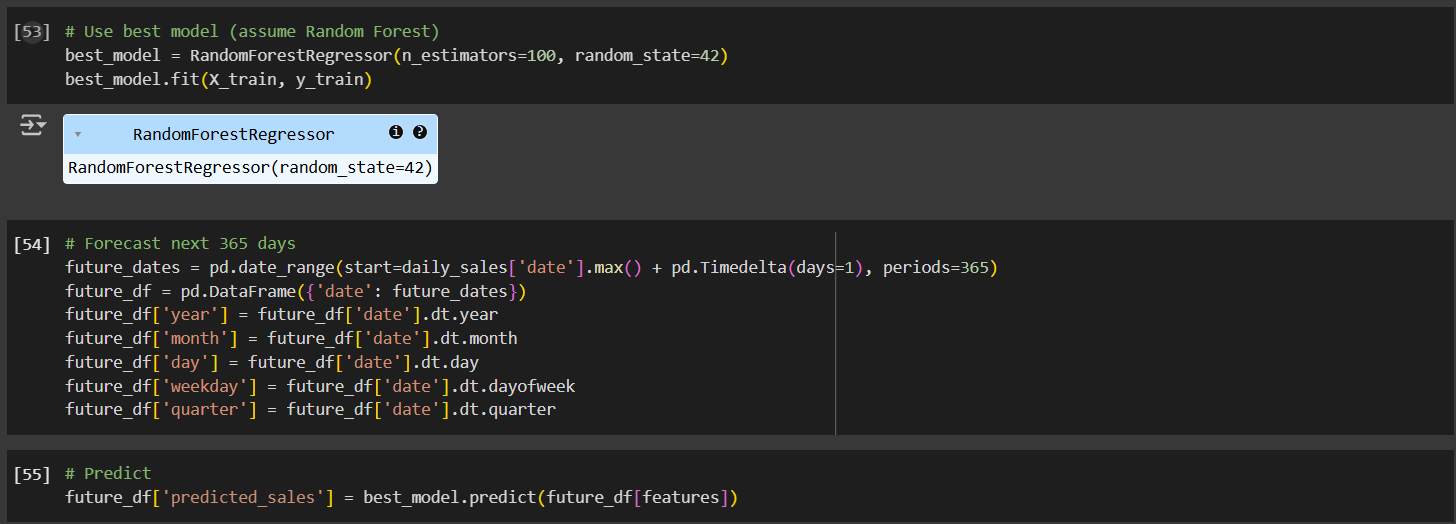
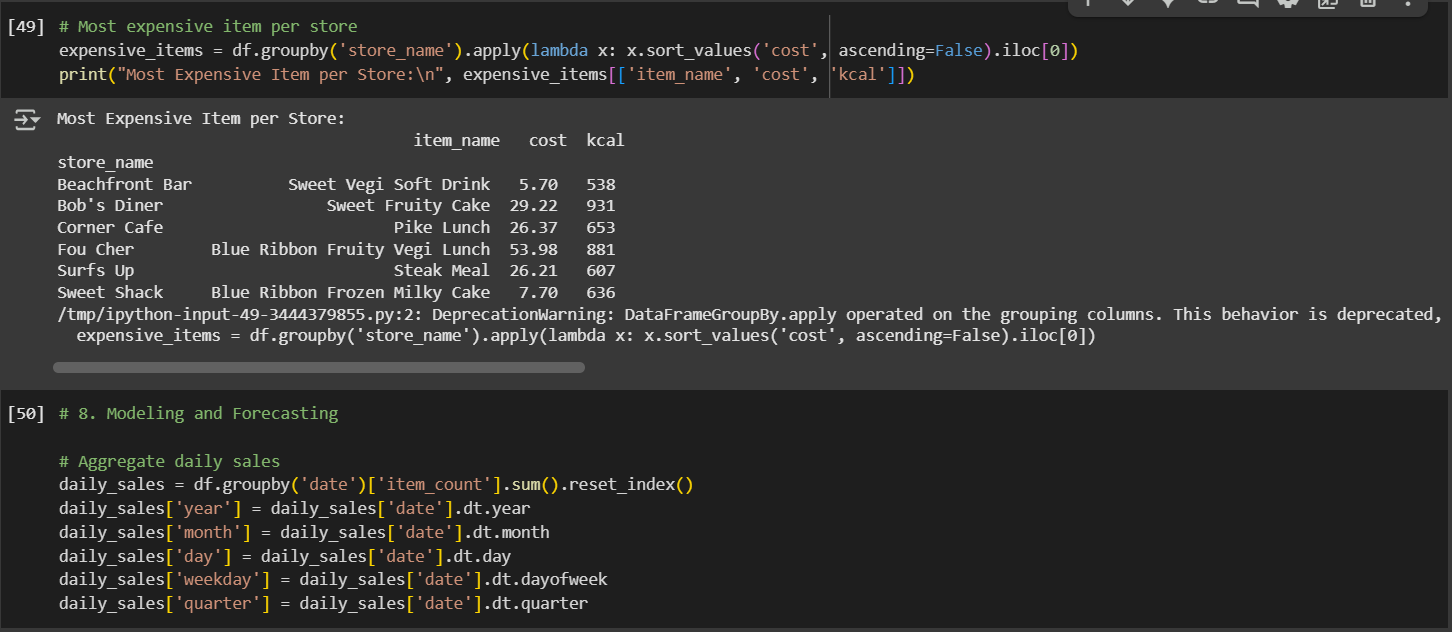
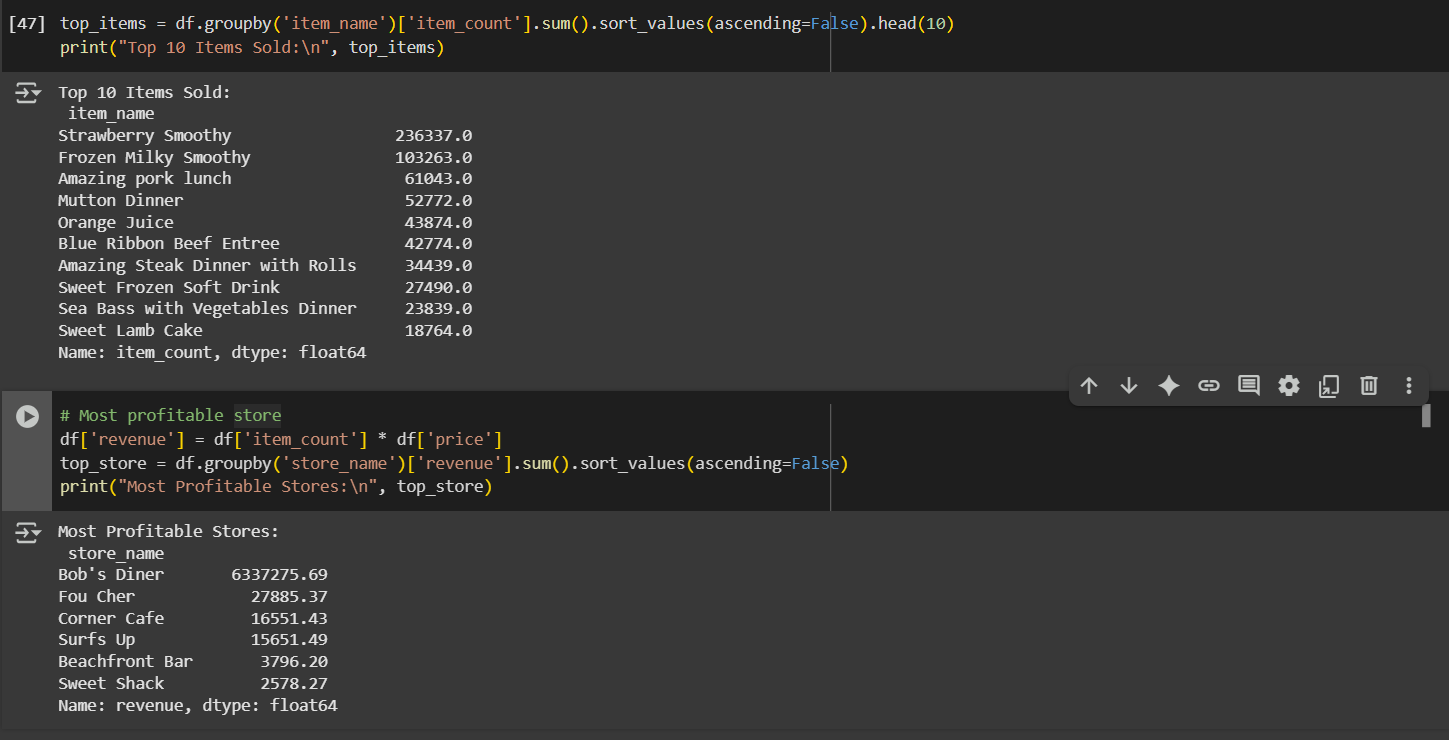
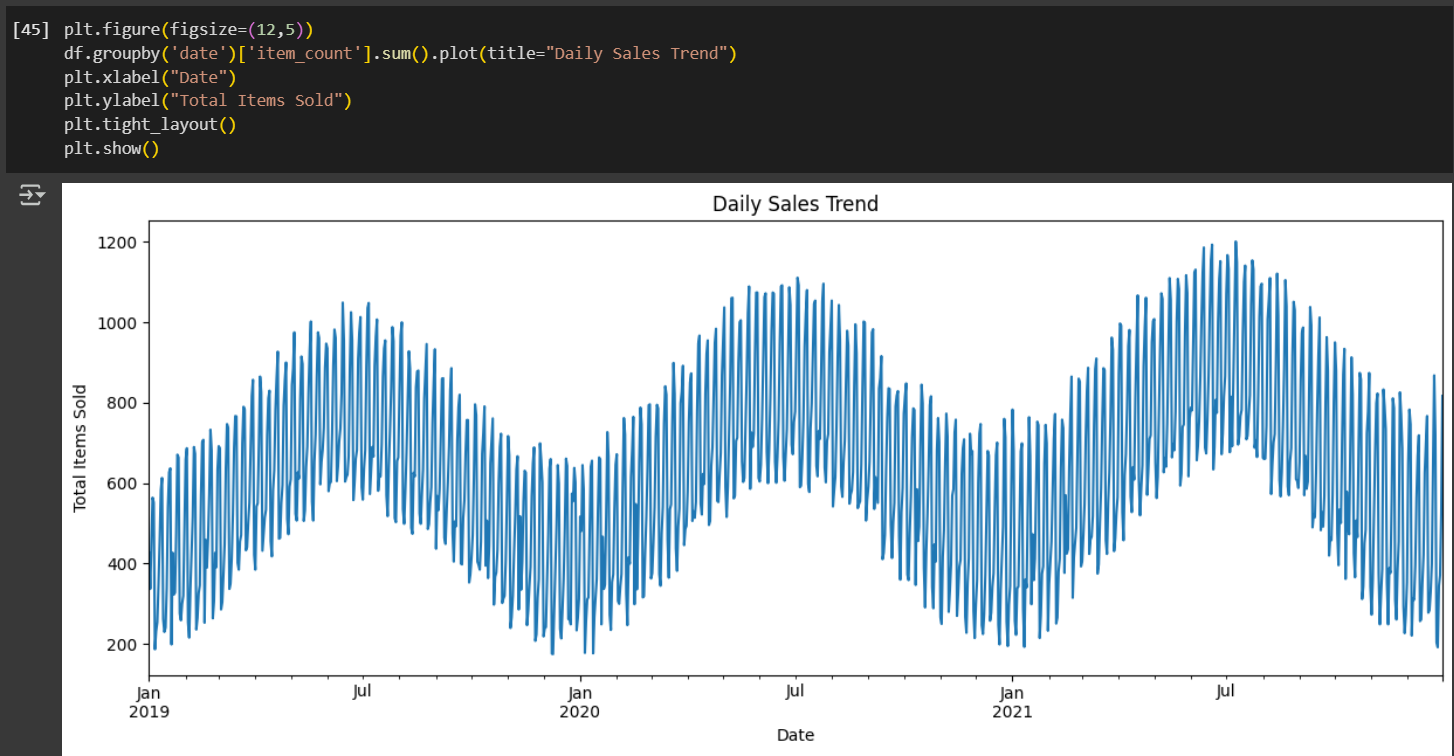
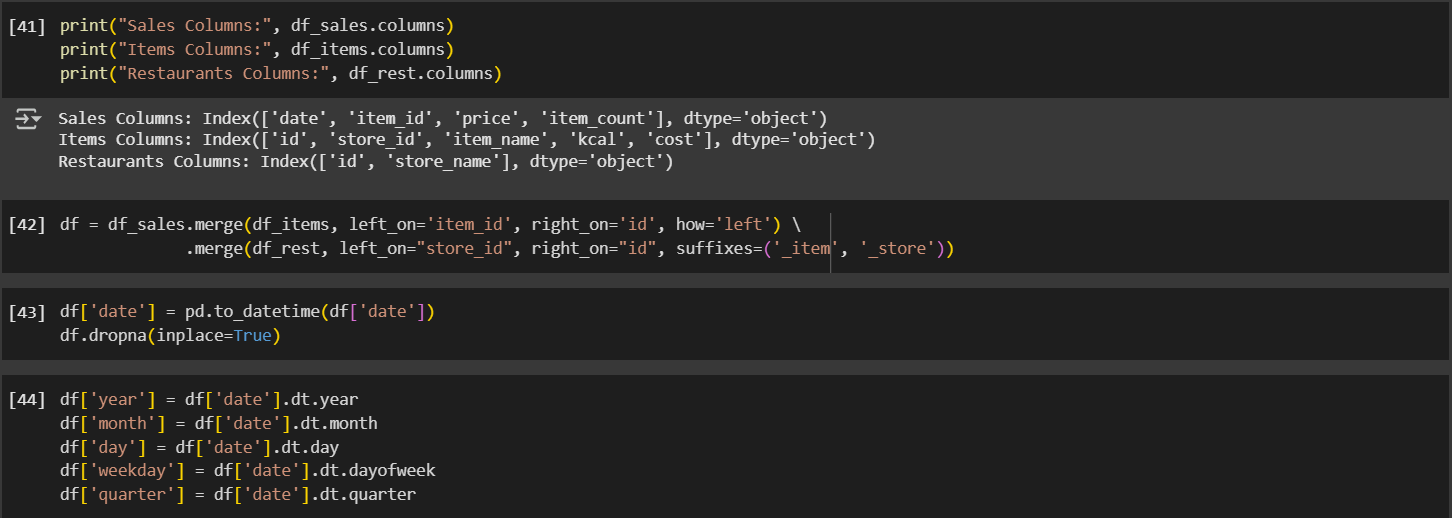
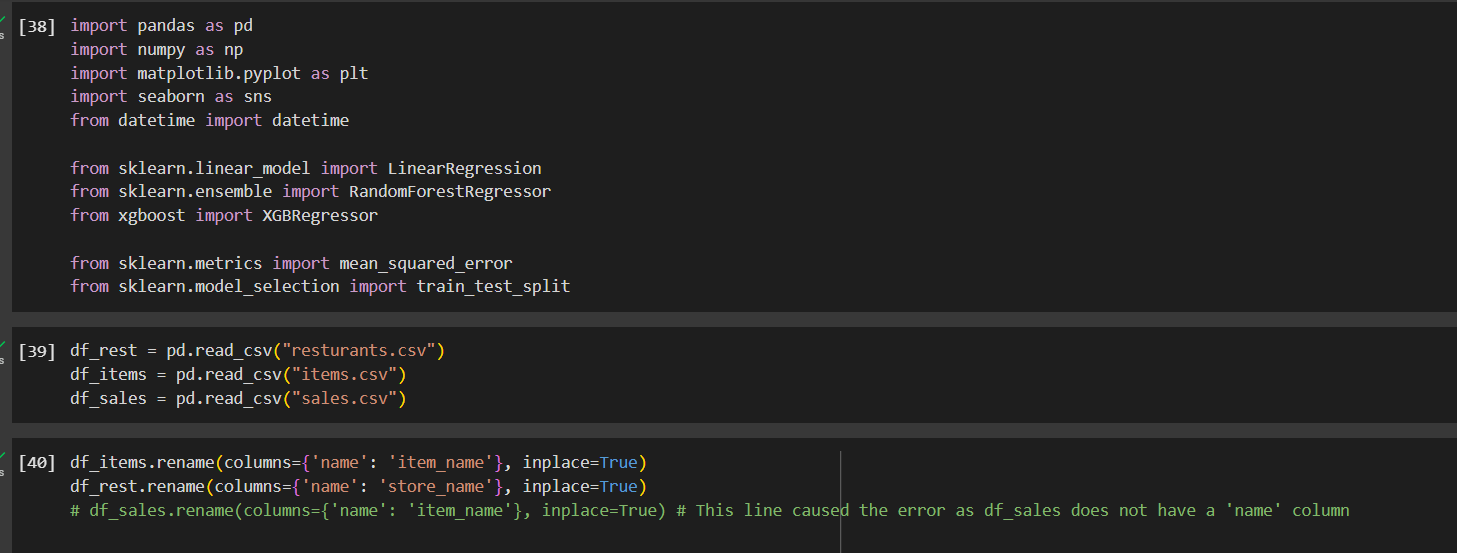
plt.ylabel("Items Sold")

plt.legend()

plt.tight\_layout()

plt.show()

**Screenshots:**

****